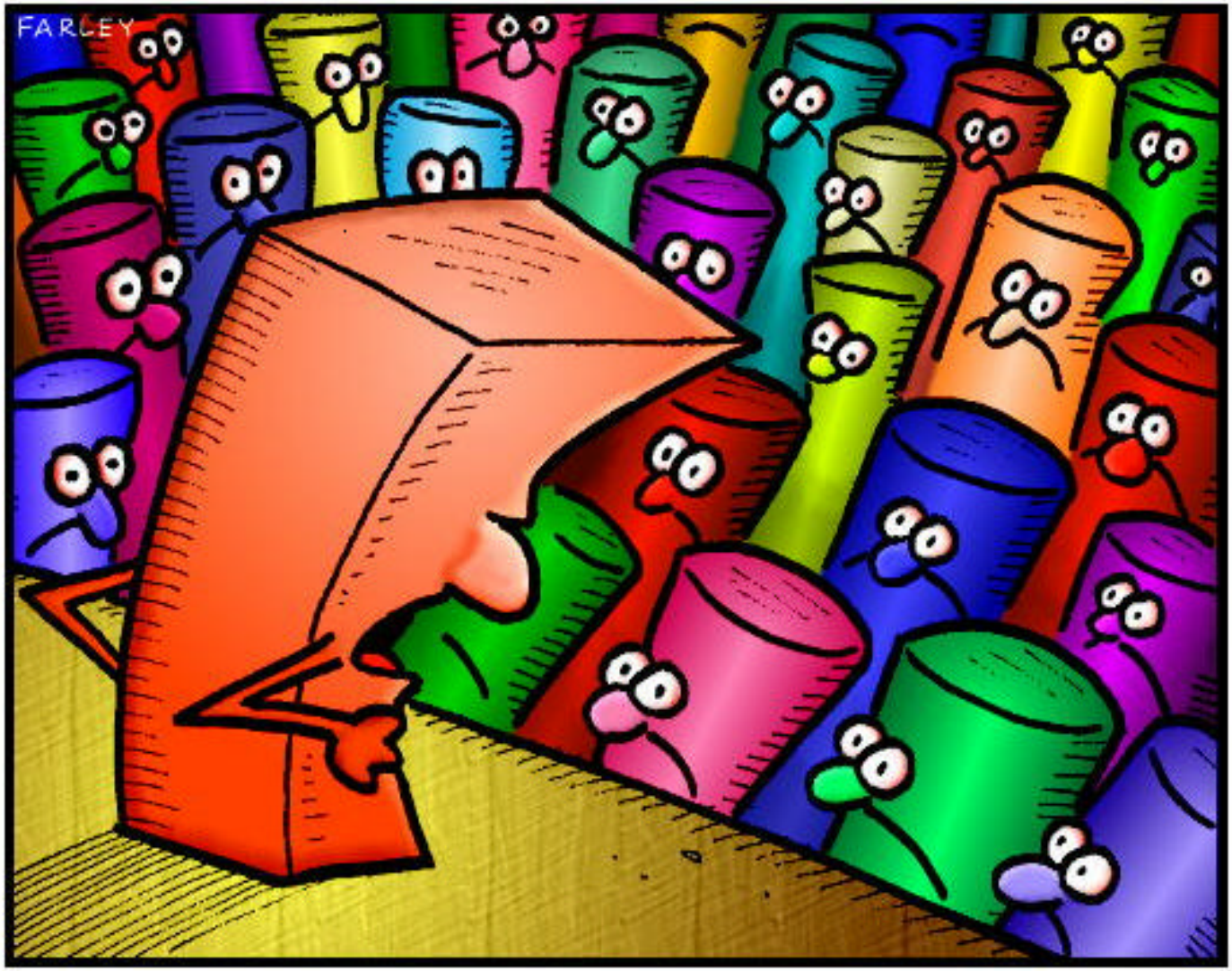
The Hidden Risks of Off-the-Shelf Integration

Engineering Square Pegs into Round Holes

Jeff Lubelczyk/522 and Dave Sames/522

DOCTOR FUN



"We turn round pegs into square pegs!"

Agenda

- Introduction
- The OTS “Mythology”
- Areas of Risk
- Risk Mitigation
- Considerations
- Case Study
- Conclusion

Introduction

- Purpose of Presentation
  - Examine areas of risk
  - Provide insight into total cost of OTS solutions
  - Identify guidelines for using OTS components
  - Share NCC SPSR real-life experiences
- Definition of OTS software
  - Software libraries, tools, or packages relating to functional aspects of the system or the development environment

Introduction (Cont'd)

- NCC SPSR
  - Replace Unisys mainframe with a workstation environment
  - Move from proprietary system to "open" system
  - Use Commercial Off-the-Shelf (COTS) software to reduce custom software maintenance costs
  - Improve software design to enhance flexibility
  - Improve NCC scheduling capabilities for increased space network utilization

Introduction (Cont'd)

- NCC SPSR
  - OTS products integrated into design: 8
  - OTS products supporting development: 21
  - Total DSI: 162k + 26k (ROSE)

The OTS “Mythology”

- Independently meets system functionality
- Plug-and-Play
- Minimal Effort
- Bug free
- Cheaper than “new” development

The Truth

- If the mythology were true, there would be no risks associated with using OTS solutions.
- However, the mythology is not true, and we must therefore identify the areas of risk and manage them appropriately.

Areas of Risk

- Schedule
- Quality
- Performance
- Maintainability
- Cost

Areas of Risk (Cont'd)

- Schedule
  - Learning curve
  - Integration
  - Unplanned-for technical problems
    - Misapplication of product
    - Bugs
    - Impacts to other development areas
  - Dependency on vendor's lifecycle
    - Turn-around time for bug fixes
    - Upgrades in functionality

Areas of Risk (Cont'd)

- Quality
  - Design driven by OTS packages
    - Decisions not necessarily correct, but because the OTS software demands them
  - Work-arounds due to OTS shortfalls
  - OTS reliability can be unknown
    - "Newness" of implementation
    - Meeting of project standards
    - Metrics
    - Testing

Areas of Risk (Cont'd)

- Performance
  - Two areas
    - Development
    - Operational
  - Tunability
  - Software size
  - Scalability

Areas of Risk (Cont'd)

- Maintainability
  - System configuration stability
  - Other tool support needed to maintain OTS products
  - Vendor support & longevity
  - Conformance to standards
  - Design consistency (nonhomogeneity)

Areas of Risk (Cont'd)

- Cost
  - All of the preceding areas of risk contribute to the overall OTS application cost
  - Cost is not just the cost of the product
    - Learn
    - Integrate
    - Maintain

Risk Mitigation

- Research
  - User groups, newsgroups, and reviews
  - Customers
  - Sales people (with salt)
- Hands-on Assessment
  - Demo version
  - Test basic design concepts

Risk Mitigation (Cont'd)

- Decoupling
  - Provide well-defined interfaces to the package
  - Isolate usage if possible
- Scheduling
  - Schedule learning time
  - Allow time for the unknown

Risk Mitigation (Cont'd)

- Expert Users
  - Cultivate experts on your team or recruit them
  - OTS Engineer
- Consultants
  - Expensive, but can be worth it
  - Review design concepts involving product
  - Debugging

Risk Mitigation (Cont'd)

- Source Code
  - Compilable in consistent manner
  - Problems can be fixed directly
  - Some isolation from software upgrades
- Replacement
  - Don't feel wedded to a product just because you bought it
  - Systems built upon a bad product are doomed to failure

Considerations

- Vendor Support & Accessibility
  - Installed user base
  - Support
  - Upgrades
  - E-mail and phone access
  - Relationship leverage
- OS Support & Licensing Agreements
  - Keeping up the “latest & greatest”

Considerations (Cont'd)

- Compatibility with other OTS packages
  - Use of other third-party products by more than one party
  - Namespace trampling
  - Industry standards
- Configuration Management Issues
  - Version control of generated code
  - Control of OTS packages against developed software releases

Considerations (Cont'd)

- CM resource requirements
- Product History
  - Past experiences
  - Limitations of product
- Documentation
  - Current
  - Accurate
  - Easy-to-use

Considerations (Cont'd)

- Cross-Platform availability
  - What's is the primary platform?
- Administrative Support
  - Periodic maintenance required
  - Training
- Time to Learn

OTS Evaluation Worksheet

	OTS Product	Product X	Product Y	Product Z
Vendor Support & Accessibility	**1.0**	8	8.5	9
Licensing	**1.0**	8	8	8
Compatibility	**1.5**	9.5	9	8.75
Configuration Management	**1.0**	5	5	5
Reliability	**1.8**	6.75	7	8.5
Integration	**1.8**	9	9	8
Scalability	**1.0**	7.5	7.5	7.5
Installation Size	**0.0**	6	7	8
Performance	**1.5**	6	7	8
Product History	**1.3**	7	7	9
Documentation	**1.0**	8.5	7	7.5
Application Context	**1.0**	5	5	5
Portability	**1.0**	8	8.5	7.5
Total Rating		9.13	9.19	9.56

The Good

- Adaptive Communications Environment
  - Provided event-framework & communications classes
  - Source code provided (20000 LOC)
  - Large users-group/support
  - Time to configure, compile, and install
  - Used (directly or indirectly) about 11000 LOC of the library
- Builder Xccessory - GUI builder
- RogueWave Libraries
  - 34 classes used with about 3500 LOC
  - Fairly robust

More Good

- Purify
  - Helped to find bugs
  - Fairly easy to use
  - Some problems with larger, complex programs
- Quantify - performance analysis
- Xemacs - editor, etc
- Perl - scripting language
- CVS - configuration management tool

The Bad

- The evolving C++ ANSI standard
  - Lack of vendor support
  - Strict compilation enforcement by aCC
- RogueWave Libraries
  - Multiple vendors use of different RW versions
  - Bugs in GUI modules
- HP Softbench - development environment
  - Difficult to use
  - Slow

More Bad

- Orbix
  - Bugs (crashes, memory leaks)
  - Overkill for application context
  - Incompatibility across implementations
- Software-through-Pictures
  - Slow
  - OS support lag
  - Scalability problems
  - SPSR-specific implementation

The Ugly

- The HP C++ compiler
  - Template instantiation
  - Code linking & symbol resolution
  - Slow

More Ugly - A Case Study

- Persistence
  - OO database objects
  - Automatic code generation
  - Automatic Oracle RDBMS table generation
  - Use StP modeling tool
  - Provide type checking at compile time

Persistence

- Actual Experience
  - Shallow learning curve
  - Poor documentation
  - Ineffective vendor support
  - Slow database generation
  - Poor mapping to RDBMS
  - Bugs
  - Poor scalability

Persistence (Cont'd)

- Impacts
  - 30 step process for DB generation (StP -> lib)
  - Integration problems
    - Slow turnaround (two weeks minimum)
    - RW usage within persistence
  - Not tunable from a RDBMS perspective
  - Large libraries leading to long-link times
  - Over-normalized database (RDBMS)

Persistence (Cont'd)

- Impacts
  - Slow performance due to objectification of data from RDBMS
  - Needed new hardware to support DB development
  - Poor design decisions due to slow turnaround time
  - Insufficient testing due to late availability of changes

Persistence (Cont'd)

- SPSR Redesign
  - After 2.5 years into a 4-year project lifecycle
  - Goals
    - Decoupling of database and application software
    - Limit use of Persistence & StP
    - Reduce size & complexity of database
    - Redesign software to support performance goals

Persistence (Cont'd)

- Conclusion
  - Negatively impacted schedule, performance, quality, maintainability, and cost
  - Cost

Purchase	$23,700.00
Licensing	$48,000.00
Development	$200,000.00
Redesign	$600,000.00
Consulting	10,000.00
Learning Curve	$150,000.00
Total:	$1,031,700.00

Persistence (Cont'd)

- What were some indicators?
  - New Product/Limited research
  - Hands-on-Assessment
    - Early use indicated scalability problems
    - Compatibility issues (RogueWave)
    - Integration problems
  - Poor vendor support
  - Poor development and runtime performance
  - Schedule slips due to database changes

Persistence (Cont'd)

- What were our options?
  - Replacement
    - Generic query interface 2-3 SM
    - Object interface: 8-9 SM
    - Advantages
      - Detailed knowledge of end product
      - Maintainable
      - Controllable
  - Bring in consultant early in development cycle

Conclusion

- Keep Risk in Mind
  - Mitigate
  - Manage
  - Especially if OTS used in critical area
- Plan for the Unknown
- Use OTS Solutions Wisely
  - Account for ALL Costs of OTS

OTS Products Purchased for SPSR Development

- Acrobat Reader
- Builder Xcessory
- ClearCase (not used)
- ClearTrack (not used)
- AR User/Notifier (CDS)
- Database Xcessory (not used)
- Eudora (not used internally)
- FrameMaker
- Netscape Gold
- Softbench (little used)
- Software through Pictures
- CVS
- Purify
- PureCoverage
- Quantify
- Xemacs
- MS Office
- MS Project
- Python
- QA Partner (not used)
- QA Agent (not used)

OTS Products Integrated Into SPSR Design

- ACE
- Orbix (removed)
- Oracle
- Ilog
- Perl
- Persistence
- RogueWave tools.h
- RogueWave view.h